Martin Eisemann,
Marcus Magnor

ZIPMAPS: Zoom-Into-Parts Texture Maps

Braunschweig : Computer Graphics Lab, 2008

Technical Report / Computer Graphics Lab, TU Braunschweig ; 2008-11-8

Veröffentlicht: 28.11.2008

http://www.digibib.tu-bs.de/?docid=00023828

Martin Eisemann
eisemann@cg.cs.tu-bs.de
Computer Graphics Lab, TU Braunschweig

Marcus Magnor
magnor@cg.tu-bs.de
Computer Graphics Lab, TU Braunschweig

ZIPMAPS: Zoom-Into-Parts Texture Maps

Technical Report 2008-11-8

November 24, 2008

Computer Graphics Lab, TU Braunschweig

Abstract

In this paper, we propose a method for rendering highly detailed close-up views of arbitrary textured surfaces. To augment the texture map locally with high-resolution information, we describe how to automatically, seamlessly merge unregistered images of different scales. Our hierarchical texture representation can easily be rendered in real-time, enabling zooming into specific texture regions to almost arbitrary magnification. Our method is useful wherever close-up renderings of specific regions shall be possible, without the need for excessively large texture maps.

Contents

Chapter 1

Introduction

In most interactive graphics applications, the scale at which some 3D object may be rendered during runtime is a-priorily unknown. For small-scale depictions, well-known mipmaps [Wil83] avoid aliasing artifacts caused by "texture minification". On the other hand, if a textured 3D object ought to be displayed at a scale larger than available texture map resolution detail-deprived, washed-out renderings are the result, due to simple interpolation. We address this problem of texture magnification.

Zoom-into-parts texture maps (zipmaps) enable rendering detailed close-up views of specific texture regions. In contrast to recent approaches like Gigapixel images [KUDC07] or clipmaps [TMJ98], we don't use a complete high-resolution texture map; instead high-resolution texture details are seamlessly merged into low-resolution textures. We show how using and rendering zipmaps is almost as simple as using standard texturemaps. The main challenge, however, is to automatically create convincing zipmaps. We address the problem of merging images at different scales into a common hierarchical texture representation, which is suited for efficient rendering in real-time, presenting a technique for creating zipmaps from a collection of unregistered photographs.

As particular contributions our paper presents:

- a new hierarchical texture mapping scheme, called zipmaps, which naturally supports enhanced magnification for specific regions.
- a fast rendering algorithm for zipmaps, which enables applying the zipmaps to arbitrary meshes in a single rendering pass. And
- a simple-to-use method to create zipmaps from a collection of photographs.

The remainder of this technical report is organized as follows. After reviewing relevant, related work in Section 2 we introduce our new zipmap textures in Section 3 and show how they are applied and efficiently rendered.

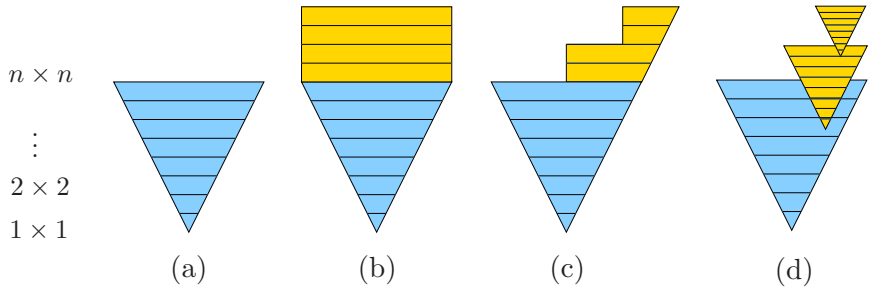


Figure 1.1: Comparison between (a) standard mipmapping – specific texture information is only provided up to a specific level; (b) clipmaps – texture information is loaded on demand; (c) multiresolution textures – a quadtree structure represents texture information at different levels; (d) our zipmaps – a sparse representation to texture specific details at higher resolution.

In Section 4 we exemplarily show how to create a zipmap texture from a collection of unregistered photographs taken by a simple consumer camera. Section 5 presents our results in detail before we discuss limitations and we conclude in Section 6.

Chapter 2

Related Work

Texture mapping was introduced in computer graphics as early as 1974 as a very effective way to increase visual rendering complexity without the need to increase geometric detail [Cat74]. To overcome the aliasing problems apparent when the texel-to-pixel ratio exceeds unity, also known as minification, Williams introduced the mipmap representation [Wil83], a precalculated image pyramid at different resolutions of the texture. Advanced variations, like ripmaps [McR98] or fipmaps [BF02], solve this problem with even higher quality, but at the cost of higher memory requirements or slower rendering. Other possibilities are summed area tables or elliptical weighted average filters. A classic survey of texture mapping can be found in [Hec86].

While the problem of texture minification is well solved, the problem of texture magnification, i.e. if the view zooms into a part of a texture, is still a very active area of research.

Interpolation: The standard and most simple approach, which is still used in most computer games due to its simplicity, is to linearily interpolate the color values between neighbouring texels. Using a nearest-neighbour approach results in blocky artefacts, while linear interpolation gives blurry results.

Super-resolution: There are probably hundreds of papers dealing with the problem of super-resolution, i.e. how to increase texture or image resolution beyond the resolution provided (in the following we will use pixels and texels interchangeably to denote single image elements). Most of these approaches are based on exemplar-images or learning-based methods which derive images statistics from either the image itself or a database of images [HJO$^+$01, SnZTyS03, HC04, YWHM08]. Other successful approaches make use of edge and gradient information or combine these with learning-based methods [FJP02, DHX$^+$07, Fat07, SXS08]. Despite good quality at moderate magnification of the images, super-resolution approaches are usually far from real-time capable and are not applicable at high magnification.

Texture Synthesis approaches create larger texture maps from one

or more small exemplar patches. One well-known approach is the image quilting technique by Efros and Freeman [EF01], in which a new image is synthesized by stitching together small patches of existing images. Kwatra *et al.* built upon this approach by using a graph cut technique to determine the optimal patch region to be used for synthesis[KSE+03]. Constrained texture synthesis tries to guide the texture creation process. The usual approach is to take neighbour information of a pixel into account and minimize some cost function which varies from approach to approach[LH05, Ram07].

For faster generation, tile-based approaches can be used. While the creation of periodic texture tiles is relatively simple, the periodicity can be annyoingly apparent for certain textures. Wang tiling can be used to allay this effect by creating patches, called Wang Tiles, which can be arranged together to non-periodically tile the plane[CSHD03, Wei04].

All these approaches only synthesize textures at a specific scale, i.e. features are usually not enlarged or shrunk in any way. In contrast Ismert *et al.*[IBG03] add detail to undersampled regions in an image-based rendering setup if more detailed versions of the same texture are available in the image. Wang and Mueller present an approach where a low resolution image guides the texture creation process for the higher resolution details [WM04]. Only recently Han *et al.* have presented an approach that uses patches at different scales for the synthesis process[HRRG08].

The problem with any of these texure synthesis approaches is that they are only suitable for textures with relatively similar repeating structures (though non-periodically arranged). The addition of specific details at specific positions is not possible. Lefebvre *et al.* presented an interactive approach to add small texture elements, called texture sprites, onto an arbitrary surface[LHN05]. While their implementation is very memory efficient and allows for various artistic effects it is less suited for rendering realistic details into an existing texture, e.g. merging two photographs.

In texture synthesis approaches the input patches are usually already aligned in a common texture space. If this is not the case registration is necessary before the stitching can be performed. A good survey on image stitching and alignment can be found in [Sze06]. Especially in the field of image-based modeling and rendering image registration is known to be a serious problem, solved with simple interpolation heuristics due to the need for real-time processing[LH96, GGSC96, BBM+01, CTMS03]. More sophisticated methods for stitching together textures on a mesh or in an image have been proposed[RCMS99, BMR01, BB01, Bau02, ADA+04, EDM+08, ZWT+05]. All these methods address the case of only relatively small registration errors, as an underlying mesh or image plane is already given in advance, plus the input images are usually of a similar scale.

Vector Textures: Texture maps are usually represented as a collection of discrete image elements and are therefore always limited in representable spatial frequency. Instead of using samples taken from the underly-

ing texture function, vector textures represent the function using resolution independent primitives. Tumblin and Choudhury save sharp boundary conditions at discrete positions for every texel to prevent some of the strong blurring apparent in usual texture magnification[TC04]. Sen uses silhouette maps to maintain sharp edges in the texture while blurring at smooth transitions [Sen04]. A similar approach which adds curved primitives was presented by Tarini and Signoni, called pinchmaps[TC05]. Finally, a complete support for all primitives of a SVG description in a vector texture was presented by Qin *et al.*[QMK08], building on their own previous work in [QMK06].

The drawback of vector textures is that they can only preserve the low and very high frequency components, while mid-frequencies and new details are not present in a close-up view. This can give vector textures a quite cartoony and unnatural look.

Large Textures: The most straight forward idea for providing detail in textures is to simply use large enough textures which are dynamically loaded on demand. But usually hardware as well as bandwidth is limited, restricting textures to be of a certain maximum size. Tanner *et al.* address this problem by introducing clipmaps[TMJ98]. In this approach the necessary data at the best matching resolution is loaded on demand depending on the viewers position. This approach works particularly well for mapping height fields[Hüt98, Los04], needed e.g. in geographic information systems (GIS). Another work in this direction are the Gigapixel images presented by Kopf *et al.*[KUDC07]. A separate thread fetches the texture tiles of the Gigapixel images needed for rendering. The authors also describe how to create these Gigapixel images from a collection of photographs. Their acquisition system must be very controlled, i.e. the camera position is fixed and images are captured on a grid. Relative position between photos is known in advance, but it takes 30 to 90 minutes to capture a complete image (even though this time can be reduced if only a low dynamic range image would be captured).

In all these approaches only scenes are considered where the needed data is in direct relation to the current viewpoint, which makes texture prefetching possible because the needed data does not change abruptly. However, this is not always the case in general texture mapping applications.

Multiresolution and Compressed Textures: Multiresolution and multiscale textures represent textures by using a hierarchichal representation. They most resemble our work presented in this paper. In the early days hierarchical texture representations were mostly used for multiresolution paint programs[BBS94, PV95] where wavelet or bandpass representations are used in a quadtree representation created on demand. Finkelstein *et al.* use binary trees of quadtrees to encode multiresolution video[FJS96]. Related to our work is the approach by Ofek *et al.*[OSW97, OSRW97] and Mayer *et al.*[MBB+01], who create a quadtree texture from a series of pho-

tographs. However these papers only describe how to merge the photos into a common texture space, based on the projection reliability and a corresponding weighting of the colour samples. They do not address the problem of how to register the images itself, which can be the main obstacle, plus the proposed data structure is not optimized for rendering. Kraus and Ertl divide an already given high-resolution image (or 3D or even 4D volume) into a regular grid of fixed sized blocks[KE02]. The information residing in these blocks is resampled into a common texture map, reducing the size of blocks with only little information. The grid then serves as an indirection table into the actual data during rendering. Using the same texture for all patches may however result in problems when applying mipmapping to the texture. Lefebvre and Hoppe use a compressed adaptive tree structure which allows for fast random access on current graphics hardware while achieving large reduction in memory requirements[LH07]. The input however, is again a given high-resolution image.

To overcome the need of explicit parameterization Benson and Davis introduce octree textures [BD02]. Using an octree instead of a quadtree allows for encoding the spatial relationship directly in the position in the octree. It also overcomes the problem of wasted texture space usually encountered in classic 2D texture atlases[gDGPR02, LBJS07].

For compression epitomes have been proposed[JFK03, CFJ05, WWOH08]. The epitome of an image is a miniature, condensed version of an image containing the essence of the textural and shape appearance.

Chapter 3

Zipmap Rendering

Zipmap textures can be thought of as a sparse sample representation of a large mipmap with almost arbitrary resolution, where only higher details for interesting parts of the texture are saved in separate texture patches and are drawn on top of each other during rendering (see Figure 1.1). Up to a specific level n the whole texture pyramid is saved in a base level mipmap texture, called the *root*. This way standard minification methods can be used to prevent aliasing in cases where the texels projected into image space are smaller than a single pixel. To incorporate details for specific regions during magnification, additional texture pyramids, called children, are added at specific positions, if needed in a recursive manner. Note that the base levels of these additional texture pyramids do not necessarily need to be at the highest level of the lower resolution image pyramid. This is beneficial for more efficient rendering or if the detail samples have been acquired at different time steps or from different viewpoints, as the affected portions of the parent patch are hidden behind the detail patches, as we will see later in Section 5.

The following is a description of the complete algorithm for rendering zipmaps onto arbitrary meshes. An overview of the complete process is also given in Figure 3.1. For rendering, the root and children are reassembled into a collection of ordered texture patches. Each one is associated with a specific texture matrix M_i which transforms texture coordinates from the root to the i-th child patch for lookup. Essentially, a zipmap texture is a simple collection of texture patches which are rendered in a specific order to texture an arbitrary surface. Patches containing the coarse overall information are rendered first, while child patches containing details are drawn later, on top of their parents.

Rendering: During rendering the color values C_i from all patches are acquired by multiplying the current texture coordinate provided by the application with the texture matrices of every patch separately. This transforms the texture coordinate from the root patchs coordinate system into the

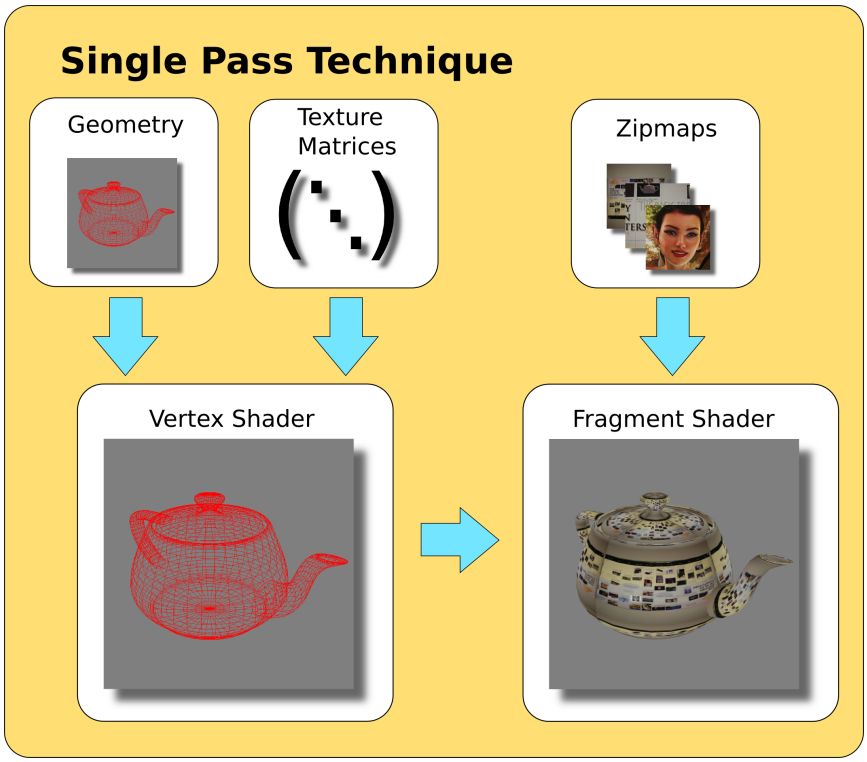


Figure 3.1: Complete overview of the rendering technique using zipmaps. Applying zipmaps is almost as simple as plain texture mapping. The incoming texture coordinates are multiplied with the zipmap texture matrices and can then be used in the fragment shader for texturing.

child coordinate system. A simple texture lookup fetches the corresponding color. The most efficient way to do this, is to do the multiplication in the vertex shader and pass the interpolated texture coordinates to the fragment shader. We then compute the final color value of the rgba-quadruple C by combining all texel rgba-values using the following simple formula:

$$C = \sum_i w_i C_i \quad , \textit{where} \tag{3.1}$$
$$w_i = \alpha_i \prod_{j>i} (1 - \alpha_j),$$

i.e. we simply mix the color value C_i of a patch with the already computed color according to the alpha channel of the patch. So in most cases a new patch is simply drawn over the old one, as most parts of the texture patches are opaque.

We can render up to 30 patches on a NVidia GeForce 8800 GTX in a single pass with this technique, because 60 floats assigned to varying variables

is the limit. If a zipmap consists of more patches, we use a slight variant of this strategy. In a first pass, the first 30 patches are drawn and written to the framebuffer as described before. Using multiple render targets, we also render the current texture coordinates of the root patch into the red and green channel of another buffer B_{tc} which is initialized to zero beforehand, and set the alpha value to one, to mark affected fragments. In the next pass, we bind the next texture patches to the texture units plus the buffer containing the texture coordinates. Now instead of rendering the whole textured mesh again, we simply draw a screen filling quad and calculate the texture coordinates of the children in the fragmentshader by making use of B_{tc}. If its alpha value is zero, we discard the fragment, keeping the old color value. Otherwise we multiply every M_i with the queried texture coordinate from B_{tc} to calculate the correct texture coordinate for the i-th patch and color the output fragment as usual. We can repeat this process until every texture patch has been processed.

Blending Patches: Current graphics hardware poses another problem whenever texture patches are drawn on top of each other. If texture values close to a patch boundary are queried, hardware interpolation will not always be able to query the correct texture value, which will create a seamless blending with the background, even if exactly the same colors are used. This is due to the employed hardware interpolation methods for border conditions which causes visible seams (Figure 3.2 left). We can easily solve this problem by applying a feathering to the alpha-channel at the border of zipmap patches (Figure 3.2 right). We do this for every level of the mipmap pyramids during the zipmap generation process, Section 4. Another advantage of this approach is that patches becoming smaller than one pixel in the output image simply disappear and do not produce small pixel artefacts that would otherwise be visible. In order to prevent drawing child patches if the calculated texture coordinates are outside the $[0 \dots 1]$ range we use hardware texture clamping. This brute force approach has proven most efficient.

Figure 3.2: Left: Close-up view with artefacts at patch borders (horizontal line in image middle). These appear even if the actual texel values are the same for the patch and the background. Right: Setting the alpha value to zero at patch boundaries removes seams.

Chapter 4

Zipmap Generation

In the following we describe an easy to use, robust approach for creating zipmaps from a collection of unregistered photographs. These photographs may even be taken from different viewpoints with different white balancing or exposure times. We arrange the image data into an exemplar tree graph (V, E) whose vertices $V = E^0, E^1, \ldots$ are the input photographs and whose edges, E, denote a parent-child relationship between them, where every child conveys a detail of the parent in a higher resolution, but not necessarily taken from the same perspective. This resembles the exemplar graph in [HRRG08], but we do not assume the scaling factor to be given.

We will describe our algorithm for a single parent-child node relationship in the following. An overview is also given in Figure 4.1. This procedure is then repeated in a top-down manner for every node of the tree.

Alignment: In a first step we use a feature based alignment technique to establish a homography between the parent and child image, I_p and I_c, assuming a planar geometry for the detail image. In most cases this is a sufficient approximation, even though we disregard parallax effects at this

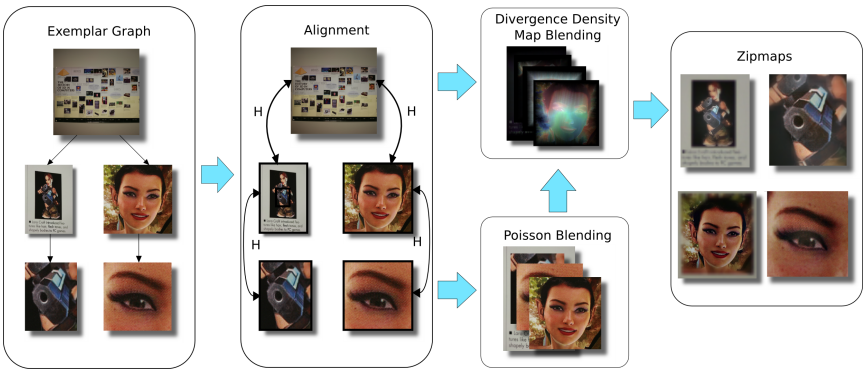


Figure 4.1: Overview of our algorithm for zipmap creation. See text for details.

point. We use the SIFT keypoint detector[Low04] because of its invariance to image transformations, especially the scale invariance is important to us, and because it provides a local descriptor for each keypoint in addition to its position. Next, we match keypoint descriptors of both images using a nearest neighbour search. Matches are rejected if the ratio between the best and second best match is below a certain threshold δ (for all our examples we set $\delta = 0.49$). In the next step we robustly estimate a homography H between the two images using RANSAC[FB81]. In each iteration we estimate a homography using the DLT algorithm[HZ06] and keep the one with the highest consensus in a least-squares sense. We can then use this matrix to warp I_c into the image space of I_p. To prevent loss of detail in I_c, we multiply the homography matrix H beforehand with a uniform scaling matrix S where the scaling factor is the largest ratio between the length of all image borders in I_c and the length of all warped image borders using H. We can then generate the warped image I_c^w through applying the scaled homography matrix to I_c Even though it might seem more natural to actually warp the images the other way round to keep all detail in I_c, we found that we can achieve better results this way, as the texel orientation during rendering is the same for all patches, which might otherwise be distracting in a closup view. Using the homography matrix we can compute the bounding rectangle in I_p, which completely contains I_c^w. This region is then extracted and also warped into the image space of I_c^w using S, creating the scaled parent region I_p^w. We use I_c^w and I_p^w in the following to create our final zipmap patch, which is used for rendering. From the bounding rectangle we can also extract the texture matrix needed for correct rendering of the patch, which comprises of a simple scaling and translation.

Color Adjustment and Blending: Different white balancing and exposure settings might cause color aberations between I_c^w and I_p^w. To adjust the colors of I_c^w to those of I_p^w we solve the constrained poisson equation

$$\begin{aligned} \Delta I^w_{poisson} &= \Delta I^w_c \\ I^w_{poisson}(x,y) &= I^w_p(x,y) \quad \text{, if } x \text{ or } y \text{ lies on a border pixel} \end{aligned} \tag{4.1}$$

, where Δ is the Laplace operator and x and y are pixel positions[PGB03]. A comparison with and without poisson blending is given in Figure 4.2.

The poisson blending resolves color changes but not the abrupt change in spatial frequency between the two images, as one is a detail of the other. We therefore apply an additional blending step between I_p^w and $I^w_{poisson}$ afterwards. An easy way to do this would be to compute a distance map or grassfire transform[Sze06], which computes the distance of every pixel to the image border or the center pixel. Since this weighted average does not take image content into account, we adopt another technique. We found that better transitions can be created if the blending is done slowly in regions

Figure 4.2: Left: Without poisson blending, the detail child patch might differ in color from the low resolution parent image. Right: After poisson blending the colors are adjusted.

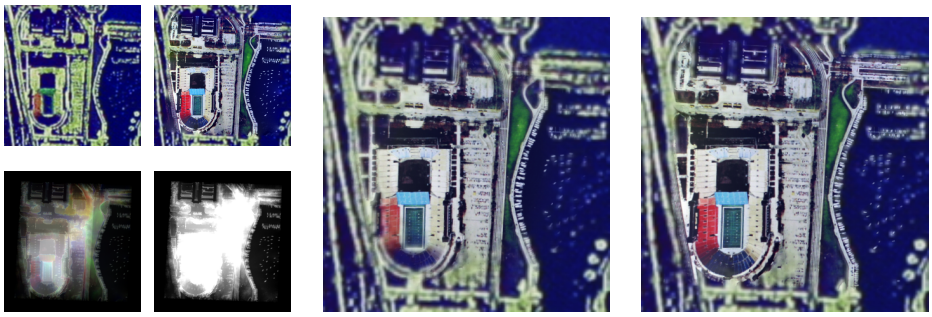
with low frequencies and faster at edges in the detail image using the colors from I_p^w at the borders and $I_{poisson}^w$ closer to the center of the image. To take this into account we make use of what we call a *gradient density map* I_{gdm}. We first establish a gradient map I_g computing for every pixel:

$$I_g = ||\nabla I_{poisson}^w||_1 = |I_{poisson,x}^w| + |I_{poisson,y}^w|, \tag{4.2}$$

where I_x and I_y are the gradients in x and y direction respectively. Using dynamic programming we then search for every pixel in I_g the least cost path in change of divergence from the pixel position to one of the border pixels and save the result in I_{gdm}, an example can be seen in Figure 4.3 (a) at the lower left. So regions with only few color gradient changes will be assigned a relatively slowly growing value from the border to the pixel of interest, while in regions with strong edges the cost value will rise faster. In addition pixels closer to the patch center will usually receive higher weights, than those close to the border. The blending is done by using a combined thresholding and blending:

$$\begin{aligned} \beta &= min(1.0, \frac{I_{gdm}}{\tau}) \\ I_r &= \beta I_{poisson}^w + (1-\beta) I_p^w, \end{aligned} \tag{4.3}$$

where τ is a user-defined threshold. The choice of τ is dependent on the patch size and the amount of gradient change. An example of this blending technique is given in Figure 4.3. We can then use I_r as the input to the next hierarchical level, if needed. If all input images have been processed, we apply the formerly mentioned feathering to I_r in a last step to acquire the final zipmaps.



(a) (b) (c)

Figure 4.3: Blending with the gradient density map. (a) Top left: Low resolution image. Top right: High resolution image. Lower left : Visualization of the gradient density map. Lower right: Corresponding alpha matte used for blending (scaled for visualization). (b) Blending result with a simple feathering. (c) Our blending result using the alpha matte from (a), the arc in the lower left is much better preserved than with simple feathering.

Chapter 5

Results and Discussion

Rendering times with zipmaps are almost the same as in standard texturing, since only a single matrix multiplication and one texture lookup per patch is required. The memory requirements are in direct accordance to the number and size of the input images used. No additional information than the patches and their 4×4 texture matrices need to be saved. Since the child patches are saved in relation to the root patch, the application programmer only has to define texture coordinates for the root patch, just as he would do with a conventional 2D texture.

As test data, we have taken input images with a handheld camera. We cannot point out exact scaling differences between the input images. However, we could robustly estimate the homographies for an approximate scaling factor of up to 12 (e.g. in the poster scene in Figure 5.1). To allow for arbitrary scaling factors we can make use of intermediate images, which are used only for the homography estimation and are not used as zipmaps. Figures 5.1 to 5.3 show results of zipmap rendering. The generation time using our non-optimized code takes about 5 minutes for a zipmap created from 4 input images at a resolution of 512×512. But many of the computations could be done on the GPU which would reduce the computation time to just a fraction of our current timings.

On the top left, the input patches are shown. On the right the zipmap texture is applied to different geometries, and some close-up views from different viewpoints and different distances are shown. The output screen resolution was always set to 1024×1024 pixels, so magnification is present in most views. Our zipmap textures can be easily applied to any kind of geometry. In Figure 5.1 we use a four patch zipmap to texture a teapot. In Figure 5.3 we apply a zipmap consisting of six patches to a simple quad for illustration purposes. On the right, some close-up views are shown. Zooming onto the knot-hole is possible without visible artefacts.

Figure 5.2 shows an interesting showcase example. The zipmap is constructed from only four input photographs. Due to the large depth conveyed

in the scene and the accompanying strong parallax effects, our algorithm cannot reconstruct a perfect zipmap. Nevertheless, we can faithfully render a plausible scene where the artefacts are almost unnoticable if the geometry or viewpoint is moved. Another interesting thing is that there are temporal differences in the input images as well, as they have been recorded at different moments in time. Even though the water fountain changes quite a lot during the acquisition no temporal artefacts are visible because the high resolution patch is always drawn on top of the low resolution images. While we did not encounter all too serious problems when registering and compositing the images together, it is quite obvious that strong parallex effects can lead to visible artefacts. In some cases the poisson blending can give results that seem unnatural to the human eye. In these cases it is helpful to add constraints to the color channels of the low resolution image in order to preserve its colors and to use only intensity values from the high resolution patch.

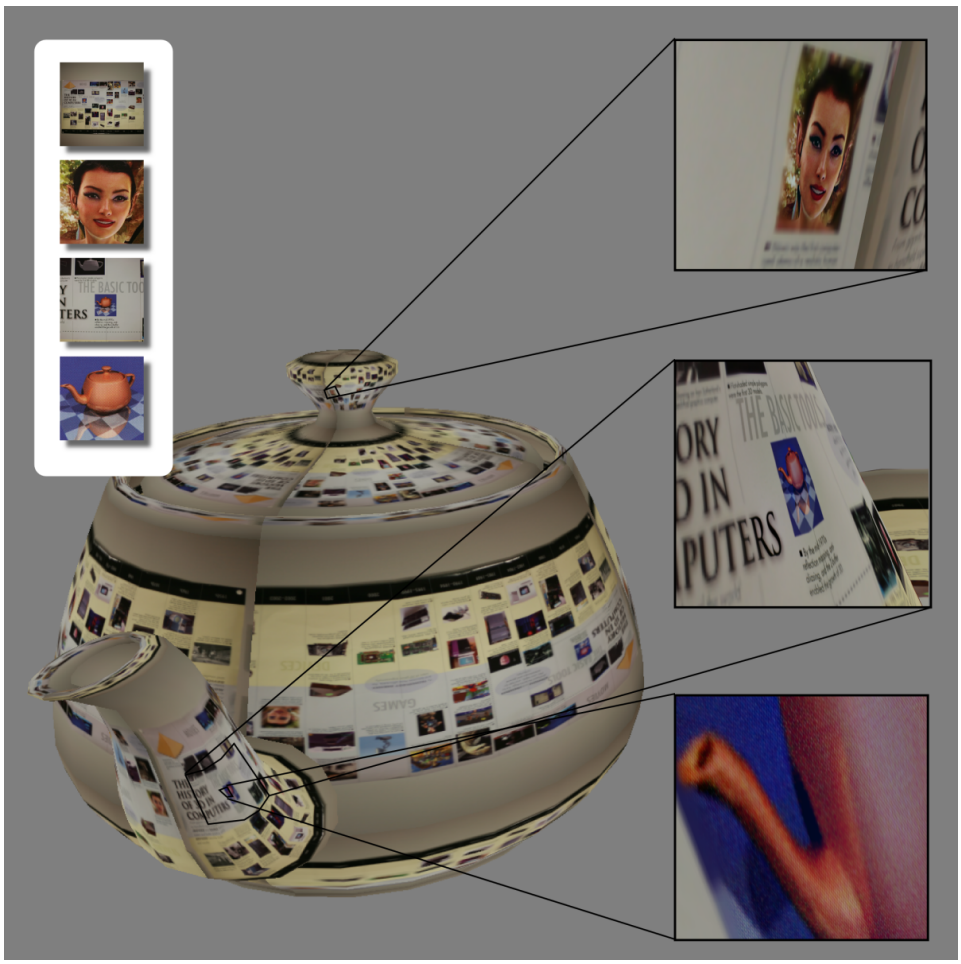


Figure 5.1: Zipmap textures can be easily applied to any geometry just like conventional textures.

Figure 5.2: Zipmap of a facade with fountain. Without 3D geometry ghosting artefacts may occur (a). Time-varying parts of the scene are merged into the common representation.

Figure 5.3: A zipmap texture acquired from six photographs and applied to a simple quad.

Chapter 6

Conclusions and Future Work

We have introduced the new concept of zipmaps, a method for rendering detailed close-up views of textured surfaces. Zipmaps are easy to use and efficient to render. We presented a simple method for generating zipmaps from a collection of unregistered photographs. Our method can be used with arbitrary images.

For future work we are planning to automatically create zipmaps from a collection of images, e.g. from a Flickr archive, where the user simply chooses the root patch. Animated zipmaps for video applications may also be interesting. Finally applying zipmaps to image-based rendering techniques like the Unwrap Mosaics[RAKRF08] will open up other new intriguing possibilities.

Bibliography

[ADA+04] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *Proc. SIGGRAPH '04*, 23(3):294–302, 2004.

[Bau02] Adam Baumberg. Blending images for texturing 3d models. In Paul L. Rosin and A. David Marshall, editors, *Proc. of the British Machine Vision Conference*, pages 404–413, 2002.

[BB01] Alexander Bornik and Joachim Bauer. High quality texture reconstruction from multiple views. *Journal of Visualisation and Computer Animation*, 12:263–276, 2001.

[BBM+01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured Lumigraph Rendering. *Proc. SIGGRAPH '01*, 20(3):425–432, 2001.

[BBS94] Deborah F. Berman, Jason T. Bartell, and David H. Salesin. Multiresolution painting and compositing. *Proc. SIGGRAPH '94*, 13(3):85–90, 1994.

[BD02] David Benson and Joel Davis. Octree textures. *Proc. SIGGRAPH '02*, 21(3):785–790, 2002.

[BF02] A. Bornik and A. Ferko. Texture minification using quad-trees and fipmaps. In *Eurographics 2002 Short Presentations*, pages 263–272, 2002.

[BMR01] Fausto Bernardini, Ioana M. Martin, and Holly Rushmeier. High-quality texture reconstruction from multiple scans. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):318–332, 2001.

[Cat74] E.E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Departement of Computer Sciences, University of Utah, 1974.

[CFJ05] Vincent Cheung, Brendan J. Frey, and Nebojsa Jojic. Video epitomes. In *CVPR '05: Proc. of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 42–49. IEEE Computer Society, 2005.

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *Proc. SIGGRAPH '03*, 22(3):287–294, 2003.

[CTMS03] J. Carranza, C. Theobalt, M. Magnor, and H. P. Seidel. Free-viewpoint video of human actors. *Proc. SIGGRAPH '03*, 22(3):569–577, 2003.

[DHX+07] Shengyang Dai, Mei Han, Wei Xu, Ying Wu, and Yihong Gong. Soft edge smoothness prior for alpha channel super resolution. In *CVPR '07: Proc. of the 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[EDM+08] Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson de Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. Floating Textures. *Computer Graphics Forum (Proc. Eurographics EG'08)*, 27(2):409–418, 2008.

[EF01] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. *Proc. SIGGRAPH '01*, 20(3):341–346, 2001.

[Fat07] Raanan Fattal. Image upsampling via imposed edge statistics. *Proc. SIGGRAPH '07*, 26(3):95, 2007.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications ACM*, 24(6):381–395, 1981.

[FJP02] William T. Freeman, Thouis R. Jones, and Egon C. Pasztor. Example-based super-resolution. *IEEE Comput. Graph. Appl.*, 22(2):56–65, 2002.

[FJS96] Adam Finkelstein, Charles E. Jacobs, and David H. Salesin. Multiresolution video. *Proc. SIGGRAPH '96*, 15(3):281–290, 1996.

[gDGPR02] David (grue) DeBry, Jonathan Gibbs, Devorah DeLeon Petty, and Nate Robins. Painting and rendering textures on unparameterized models. *Proc. SIGGRAPH*, 21(3):763–768, 2002.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. *Proc. SIGGRAPH '00*, 19(3):43–54, 1996.

[HC04] Y.M. Xiong H. Chang, D.-Y. Yeung. Super-resolution through neighbor embedding. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*, volume 1, pages 275–282, 2004.

[Hec86] Paul S. Heckbert. Survey of Texture Mapping. *IEEE Comput. Graph. Appl.*, 6(11):56–67, 1986.

[HJO+01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. *Proc. SIGGRAPH '01*, 20(3):327–340, 2001.

[HRRG08] Charles Han, Eric Risser, Ravi Ramamoorthi, and Eitan Grinspun. Multiscale texture synthesis. *Proc. SIGGRAPH '08*, 27(3):1–8, 2008.

[Hüt98] Tobias Hüttner. High resolution textures. In *Visualization '98*, pages 13–17, 1998.

[HZ06] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2006.

[IBG03] Ryan M. Ismert, Kavita Bala, and Donald P. Greenberg. Detail synthesis for image-based texturing. In *I3D '03: Proc. of the 2003 symposium on Interactive 3D graphics*, pages 171–175. ACM, 2003.

[JFK03] Nebojsa Jojic, Brendan J. Frey, and Anitha Kannan. Epitomic analysis of appearance and shape. In *Proc. ICCV*, pages 34–41, 2003.

[KE02] Martin Kraus and Thomas Ertl. Adaptive texture maps. In *HWWS '02: Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 7–15. Eurographics Association, 2002.

[KSE+03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *Proc. SIGGRPAH '03*, 22(3):277–286, 2003.

[KUDC07] Johannes Kopf, Matt Uyttendaele, Oliver Deussen, and Michael F. Cohen. Capturing and viewing gigapixel images. *Proc. SIGGRPAH '07*, 26(3):93, 2007.

[LBJS07] Julien Lacoste, Tamy Boubekeur, Bruno Jobard, and Christophe Schlick. Appearance preserving octree-textures. In *GRAPHITE '07: Proc. of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 87–93. ACM, 2007.

[LH96] Marc Levoy and Pat Hanrahan. Light Field Rendering. *Proc. SIGGRAPH '96*, 15(3):31–42, 1996.

[LH05] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *Proc. SIGGRAPH '05*, 24(3):777–786, 2005.

[LH07] Sylvain Lefebvre and Hugues Hoppe. Compressed random-access trees for spatially coherent data. In *Rendering Techniques*, pages 339–349, 2007.

[LHN05] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Texture sprites: Texture elements splatted on surfaces. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)*, pages 163–170. ACM Press, 2005.

[Los04] Frank Losasso. Geometry clipmaps: terrain rendering using nested regular grids. *Proc. SIGGRAPH '04*, 23(3):769–776, 2004.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[MBB+01] Heinz Mayer, Alexander Bornik, Joachim Bauer, Konrad Karner, and Franz Leberl. Multiresolution texture for photorealistic rendering. In *In Proc. 17 th Spring Conference on Computer Graphics*, pages 174–183. IEEE Computer Society, 2001.

[McR98] T. McReynolds. *Programming with Opengl: Advanced Rendering.* ACM, 1998.

[OSRW97] Eyal Ofek, Erez Shilat, Ari Rappoport, and Michael Werman. Multiresolution textures from image sequences. *IEEE Comput. Graph. Appl.*, 17(2):18–29, 1997.

[OSW97] Eyal Ofek, Erez Shilat, and Michael Werman. Highlight and reflection-independent multiresolution textures from image sequences. *IEEE Computer Graphics and Applications*, 17:18–29, 1997.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *Proc. SIGGRPAH '03*, 22(3):313–318, 2003.

[PV95] Ken Perlin and Luiz Velho. Live paint: painting with procedural multiscale textures. *Proc. SIGGRAPH '95*, 14(3):153–160, 1995.

[QMK06] Zheng Qin, Michael D. McCool, and Craig S. Kaplan. Real-time texture-mapped vector glyphs. In *I3D '06: Proc. of the 2006 symposium on Interactive 3D graphics and games*, pages 125–132. ACM, 2006.

[QMK08] Zheng Qin, Michael D. McCool, and Craig Kaplan. Precise vector textures for real-time 3d rendering. In *SI3D '08: Proc. of the 2008 symposium on Interactive 3D graphics and games*, pages 199–206. ACM, 2008.

[RAKRF08] Alex Rav-Acha, Pushmeet Kohli, Carsten Rother, and Andrew Fitzgibbon. Unwrap mosaics: A new representation for video editing. *Proc. SIGGRAPH '08*, 27(3):17, 2008.

[Ram07] Ganesh Ramanarayanan. Constrained texture synthesis via energy minimization. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):167–178, 2007.

[RCMS99] Claudio Rocchini, Paolo Cignomi, Claudio Montani, and Roberto Scopigno. Multiple textures stitching and blending on 3D objects. In *Proc. of the Eurographics Workshop on Rendering*, pages 119–130, 1999.

[Sen04] Pradeep Sen. Silhouette maps for improved texture magnification. In *HWWS '04: Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 65–73. ACM, 2004.

[SnZTyS03] Jian Sun, Nan ning Zheng, Hai Tao, and Heung yeung Shum. Image hallucination with primal sketch priors. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR*, pages 729–736, 2003.

[SXS08] J. Sun, Z.B. Xu, and H.Y. Shum. Image super-resolution using gradient profile prior. In *CVPR08*, pages 1–8, 2008.

[Sze06] Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, 2006.

[TC04] Jack Tumblin and Prasun Choudhury. Bixels: Picture samples with sharp embedded boundaries. In Alexander Keller

and Henrik Wann Jensen, editors, *Rendering Techniques*, pages 255–264. Eurographics Association, 2004.

[TC05] Marco Tarini and Paolo Cignoni. Pinchmaps: Textures with customizable discontinuities. *Computer Graphics Forum*, 24(3):557–568, 2005.

[TMJ98] Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones. The clipmap: a virtual mipmap. *Proc. SIGGRAPH '98*, 17(3):151–158, 1998.

[Wei04] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 67. ACM, 2004.

[Wil83] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, 1983.

[WM04] Lujin Wang and Klaus Mueller. Generating sub-resolution detail in images and volumes using constrained texture synthesis. In *VIS '04: Proc. of the conference on Visualization '04*, pages 75–82. IEEE Computer Society, 2004.

[WWOH08] Huamin Wang, Yonatan Wexler, Eyal Ofek, and Hugues Hoppe. Factoring repeated content within and among images. *Proc. SIGGRAPH '08*, 27(3):1–10, 2008.

[YWHM08] J.C. Yang, J. Wright, T.S. Huang, and Y. Ma. Image super-resolution as sparse representation of raw image patches. In *CVPR08*, pages 1–8, 2008.

[ZWT+05] Kun Zhou, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. *Proc. SIGGRAPH '05*, 24(3):1148–1155, 2005.